Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 3935804, 10 pages
https://doi.org/10.1155/2018/3935804

Research Article

An Efficient Heuristic Algorithm for Solving Connected Vertex Cover Problem

Yongfei Zhang,[1] Jun Wu,[1] Liming Zhang,[2,3] Peng Zhao,[1] Junping Zhou,[1,2] and Minghao Yin[1,2]

[1]*College of Information Science and Technology, Northeast Normal University, Changchun 130117, China*
[2]*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, China*
[3]*College of Computer Science and Technology, Jilin University, Changchun 130012, China*

Correspondence should be addressed to Junping Zhou; zhoujp877@nenu.edu.cn

Received 26 January 2018; Revised 1 June 2018; Accepted 3 August 2018; Published 6 September 2018

Academic Editor: Haipeng Peng

The connected vertex cover (*CVC*) problem, which has many important applications, is a variant of the vertex cover problem, such as wireless network design, routing, and wavelength assignment problem. A good algorithm for the problem can help us improve engineering efficiency, cost savings, and resources consumption in industrial applications. In this work, we present an efficient algorithm GRASP-CVC (Greedy Randomized Adaptive Search Procedure for Connected Vertex Cover) for *CVC* in general graphs. The algorithm has two main phases, i.e., construction phase and local search phase. In the construction phase, to construct a high quality feasible initial solution, we design a greedy function and a restricted candidate list. In the local search phase, the configuration checking strategy is adopted to decrease the cycling problem. The experimental results demonstrate that GRASP-CVC is better than other comparison algorithms in terms of effectivity and efficiency.

1. Introduction

The connected vertex cover (CVC), which was first introduced by Garey and Johnson in paper [1], is one of the classical combinatorial optimization problems. The problem not only shows its great importance in theory, but also has many significant industrial applications [2–5]. For example, in the wireless network design, the vertices of the network are connected by transmission links. Because network signals damp with transmission, we want to place a minimum number of relay stations on vertices in order to assure that any two neighboring stations are connected and that every transmission link is connected to a relay station. This is the most direct application of the connected vertex cover model in the industry. Designing a good algorithm to solve this problem can not only improve work efficiency and save money and labor cost, but also save natural resources and reduce material waste. The problem is known to be NP-hard even in the planar 2-connected graph of maximum degree 4 [6] and planar bipartite graph with maximum degree 4 [7], as well as in 3-connected graph [8].

The *CVC* problem has been studied for a long time, and a lot of efforts have been devoted to it. To date, there are mainly two types of algorithms to solve *CVC*, i.e., exact algorithms and approximation algorithms. All existing exact algorithms for *CVC* are mainly FPT (fixed-parameter tractable) algorithms in theory and these theoretical results are obtained in worst case. For example, Moser [2] showed that *CVC* was fixed-parameter tractable using the tree width as a parameter and proposed a dynamic programming algorithm running in $O(2^w \cdot w^{3w+2} \cdot n)$ time, where w was the tree width and n was the number of nodes of nice tree decomposition. With the desired vertex cover size k as parameter, Richter et al. [9] proposed an improved algorithm with running time in $O(2.7606^k)$ in the worst case. Binkele-Raible [10] provided a better exact algorithm with running time in $O(2.4882^k)$ in the worst case. Because these exact algorithms failed to solve large graphs, a lot of efforts have been devoted to approximation algorithms. In the general graph, Savage [11] proposed the first constant ratio algorithm and proved that the set of internal nodes of any depth-first search tree was a solution of 2-approximation

for *CVC* problem. In addition, Fujito and Doi [12] proposed a 2-approximation algorithm for solving CVC, which ran in $O(\log^2 n)$ time using $O(\delta^2(m+n)/\log n)$ processors on an EREW-PRAM, where n was the number of vertices, m was the number of edges, and δ was the maximum vertex degree. Fernau and Manlove [7] proved that *CVC* was NP-hard to approximate within $10\sqrt{5}-21$ in general graphs unless $P = NP$. Therefore, it is difficult to improve the approximation ratio of approximation algorithms in general graphs, which makes the researchers change their research angle into the special graphs. Escoffier et al. [13] proved that the *CVC* problem was APX-complete in bipartite graphs of maximum degree 4 and was polynomial time solvable in chordal graphs. In addition, they also showed that CVC was 5/3-approximable for a class of special graphs (where solving the minimum vertex cover problem used polynomial time) and a polynomial time approximation algorithm for *CVC* in planar graphs was presented. Cardinal and Levy [14] proposed an approximation algorithm in dense graphs and the algorithm approximated the *CVC* problem with a ratio strictly less than 2 in dense graphs. The first polynomial time approximation algorithm in unit disk graphs for *CVC* problem was proposed in [3]. Li et al.[15] proved that the *CVC* problem was still NP-hard for 4-regular graphs and provided a lower bound for the problem. Moreover, they proposed two approximation schemes for this problem in 4-regular graphs with approximation ratio 3/2 and $4/3 + O(1/n)$, respectively. Although the exact algorithms for *CVC* can provide an optimal solution, they are hard and time consuming to deal with large scale instances. Furthermore, although some approximation algorithms for *CVC* can get good performance in special graphs, they are usually not suitable for dealing with general graphs, and the state-of-the-art approximation methods in general graphs can only provide an approximate ratio 2, which is often not enough in practice. This yields a new challenge for us to devise a heuristic algorithm for *CVC* that can deal with large general graphs and obtain the best possible approximate solutions within a reasonable time.

In this article, the heuristic algorithm GRASP-CVC for *CVC* in general graphs is proposed and this algorithm can obtain a relatively good solution within a reasonable time. The heuristic algorithm GRASP-CVC is based on the framework of greedy randomized adaptive search procedure (*GRASP*) [16]. The algorithm GRASP-CVC has two main phases, i.e., construction phase and local search phase. In the construction phase, the GRASP-CVC tries to construct a feasible initial solution greedily. During this phase, we design a greedy function to help evaluate the benefit of adding a vertex to the current solution. Besides, we construct a restricted candidate list (*RCL*) to assist in constructing a high quality initial solution in this phase. In the local search phase, the initial solution is further improved. To prevent the local search from suffering severe cycling problem, the configuration checking (*CC*) strategy is adopted in the search. Relying on the *CC*, we avoid many unnecessary searches during the local search procedure and greatly improve the efficiency of the GRASP-CVC. Once the local search phase cannot explore a better solution anymore, which means the local search phase reaches a local optima, the GRASP-CVC then restarts a new iteration and repeats the construction and local search phases until reaching the maximum iteration times. The best found solution will be the final solution after all iterations are used up. The experimental results demonstrate that GRASP-CVC is better than other comparison algorithms in terms of effectivity and efficiency. Moreover, the GRASP-CVC obtains solutions of almost the same size in 10 times of running, which demonstrates its stability.

The rest of this paper is structured as follows. Some relevant definitions and background knowledge will be introduced in the next section. In Section 3, the algorithm GRASP-CVC will be introduced and the two main components will be discussed in detail. Experimental evaluations and analyses will be shown in Section 4. Conclusions and future work will be given in the last section.

2. Preliminaries

In this section, some definitions and background knowledge are provided. From now on, unless otherwise stated, we only consider the CVC problem on an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertices set and $E = \{e_1, e_2, \ldots, e_m\}$ is the edges set. In addition, each edge $e_i = (v_k, v_j)(1 \leqslant i \leqslant m, 1 \leqslant k, j \leqslant n, k \neq j)$ is a 2-element tuple on V and we define vertices v_k and v_j as the endpoints of edge e_i. For a vertex subset $C \subseteq V$ and an edge e_i, if C contains no endpoint of e_i, we say e_i is uncovered by C; otherwise, we say e_i is covered by C.

Definition 1 (vertex cover, VC). Given a graph $G = (V, E)$, a subset of vertices $C \subseteq V$ is a vertex cover (*VC*) of G if each edge in E has at least one endpoint in C.

Definition 2 (minimum vertex cover, MVC). Given a graph $G = (V, E)$, the minimum vertex cover (*MVC*) problem is to compute a *VC* of minimum cardinality in G.

Definition 3 (induced subgraph, IS). Given two graphs $G = (V, E)$ and $G' = (V', E')$, where $V' \subseteq V$ and $E' = \{(v, u) \mid v, u \in V' \wedge (v, u) \in E\}$, G' is called an induced subgraph (IS) of G.

Definition 4 (connected graph). A graph is a connected graph if there is a path between every pair of vertices.

Definition 5 (connected vertex cover, CVC). Given a connected graph $G = (V, E)$, the connected vertex cover (*CVC*) problem is to determine a subset $C \subseteq V$ with minimum cardinality such that C satisfies the following two conditions: (1) C is a vertex cover; (2) the IS induced by C is a connected graph.

In order to help the readers to understand the concepts given above, we provide an example in Figure 1. Figure 1(a) is a graph $G = (V, E)$, where $V = \{a, b, c, d, e, f, g\}$ and $E = \{(a, c), (b, c), (c, d), (d, e), (e, f), (e, g)\}$. Figure 1(b) presents an induced subgraph by the vertex set $V_1' = \{c, e\}$ that is just the solution of the *MVC* problem of G. Because the induced

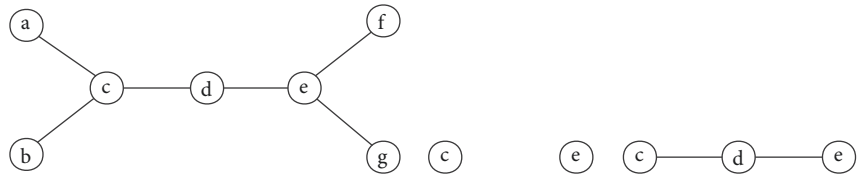

Figure 1: An example for MVC and CVC.

subgraph in Figure 1(b) is not connected, the vertex set $V_1' = \{c, e\}$ is not the solution of the *CVC* problem of G. By adding a vertex d to V_1', we obtain a new vertex set $V_2' = \{c, d, e\}$. The subgraph induced by V_2' is shown in Figure 1(c). From Figure 1(c), we can notice that the vertex set V_2' is the minimal vertex subset that satisfies the following: (1) subgraph induced by V_2' is connected; (2) the vertex set V_2' covers all edges in E. Thus, V_2' is a solution for the *CVC* problem of G. From the example, we see that the size of the optimal *MVC* solution provides a lower bound for the size of the optimal *CVC* solution. In the following, we will present the conclusion in Theorem 6.

Theorem 6. *The size of the optimal MVC solution provides a lower bound for the size of the optimal CVC solution.*

Proof. Given an undirected graph G, we suppose the size of the optimal *MVC* solution is N and the size of the optimal *CVC* solution is M. Then we will analyze the theorem cases individually.

Case 1. There is a connected optimal solution of *MVC*. Under this circumstance, the *MVC* solution is also a *CVC* solution. Thus, we have $M = N$.

Case 2. There is no connected solution among all of the optimal solutions of *MVC*. Under this condition, it is impossible that $M = N$. Then we will prove the conclusion by using reduction to absurdity. Suppose there exists a *CVC* solution that $M \leqslant N$. Under this condition, according to the definitions of *CVC* and *MVC*, we know the *CVC* solution is an *MVC* solution as well. So, we get one *MVC* solution whose size is smaller than N. However, this is a contradiction with the previous assumption that N is the size of the optimal *MVC* solution.

In total, we finally reach the conclusion that $M \geqslant N$, which means that the optimal *MVC* solution size provides a lower bound for the size of the optimal *CVC* solution. □

3. GRASP-CVC Algorithm for CVC

GRASP-CVC (Greedy Randomized Adaptive Search Procedures for Connected Vertex Cover problem) is a multistart metaheuristic, which consists of two main phases: construction phase and local search phase. An initial solution is constructed firstly in the construction phase, and then the local search phase attempts to find the existence of a better solution by exploring the neighborhood of the initial solution. The two phases are executed repeatedly until reaching the termination condition, and then the GRASP-CVC takes the best found solution as the final output. The pseudocode of GRASP-CVC is outlined in Algorithm 1. At first, the solution C^* is initialized (line 1). Then the algorithm enters an iteration loop (lines 2–8). In each iteration, an initial *CVC* solution C is generated firstly by the construction procedure (line 3), and then in the local search procedure (line 4), GRASP-CVC starts its search from C trying to find a better solution. If C possesses fewer vertices than the current best solution C^*, C^* will be updated by C (lines 5-6). When the GRASP-CVC reaches the maximum iteration times, C^* will be returned as the final solution (line 9). During the construction, we design a greedy function and construct a restricted candidate list to help to construct a high quality feasible solution. Moreover, in the local search, we adopt the configuration checking to reduce the cycling problem. The two phases will be discussed in detail in the next two subsections.

```
1 initialize the solution C*;
2 for i = 1 to MaxIter do
3     C = GreedyConstruction(seed);
4     C = LocalSearch(C);
5     if |C| < |C*| then
6         C* = C;
7     end
8 end
9 return C*
```

Algorithm 1: GRASP-CVC(*MaxIter*, *seed*).

3.1. Construction Phase. Before introducing the construction phase, we shall give the definitions of greedy function and restricted candidate list (*RCL*) that play important roles in the construction phase.

3.1.1. Greedy Function and RCL. To evaluate the benefit of adding v to the current solution, a greedy function $score(v)$ is designed, which is quite important for the construction of *RCL* as well. In order to introduce the greedy function, we firstly propose some relevant definitions. We say vertices u and v are neighbors of each other if there is an edge between them, and we use $N(u) = \{v \mid (u, v) \in E\}$ denoting the neighbor set of vertex u. The neighbor vertices set of a solution C, denoted as $N(C)$, can be calculated as follows:

```
1  C = Φ;
2  RCL = {v | v ∈ V ∧ score(v) ≥ score(u), ∀u ∈ V};
3  initialize the score of each vertex according to Formula(2);
4  while C is not a connected vertex cover do
5     choose a vertex v from RCL randomly;
6     C = {v} ∪ C;
7     update score of each vertex;
8     RCL = {v | v ∈ N(C) ∧ score(v) ⩾ score(u), ∀u ∈ N(C)};
9  end
10 return C
```

Algorithm 2: GreedyConstruction(*seed*).

$$N(C) = \{v \mid v \notin C,\ v \in N(u),\ u \in C,\ N(u) = \{v \mid (u,v) \in E\}\} \tag{1}$$

For a given vertex v, the greedy function $score(v)$ can be calculated by

$$score(v) = cost(C) - cost\left(C'\right) \tag{2}$$

In Formula (2), C is the current solution. If v is in C, $C' = C \setminus \{v\}$ ("\" means removing the vertex v from C); otherwise, $C' = C \cup \{v\}$. Moreover, $cost(C)$ is also a function calculated by

$$cost(C) = \left|\{e \mid e \text{ is not covered by } C\}\right| \tag{3}$$

From the formula, we can know that the function $cost(C)$ is to compute the total number of edges uncovered by C. In addition, when v is in C, $score(v)$ is a negative number.

Using the greedy function $score(v)$, we can identify those vertices that are most beneficial to the current solution. Moreover, the greedy function is an indispensable part in the construction of *RCL*.

The *RCL* consists of the vertices that are most beneficial to the current solution C. In the construction phase, one vertex is chosen randomly from *RCL*. And to construct a feasible initial solution, we select vertices from *RCL*. The construction of *RCL* is described in

$$RCL = \{v \mid v \in N(C) \wedge score(v) \geqslant score(u),\ \forall u \in N(C)\} \tag{4}$$

Clearly, the elements of *RCL* are the vertices set having the highest *score* in the premise of not destroying the connectivity of C.

3.1.2. Construction Procedure. After the necessary descriptions of greedy function and *RCL*, we shall discuss the greedy construction procedure in detail. The greedy construction procedure GreedyConstruction is outlined in Algorithm 2.

In the beginning, the connected vertex cover C is initialized (line 1) and the *score* of each vertex is initialized according to Formula (2) (line 2). The *RCL* is initialized to the vertices with the highest *score* among the vertices set V (line 3). Then, the procedure enters the main loop (lines 4–8). In each loop, a vertex v is chosen from the *RCL* randomly and added to the construction solution C (lines 5-6). After the operations of lines 5 and 6, the *score* of each vertex is updated according to Formula (2) (line 7). At the end of the loop, the *RCL* is updated (line 8). The loop is executed until C is constructed to be a solution of *CVC*, and then C will be returned at the end of the construction procedure (line 10).

3.2. Local Search Phase. In this subsection, we shall introduce the configuration checking (*CC*) strategy and discuss the working process of the local search phase in detail.

3.2.1. Configuration Checking Strategy. Greedy strategy is usually an important part in the local search algorithms. It helps to lift the performance of the local search algorithms on large and hard instances. However, the greedy strategy usually also makes the local search algorithms easier to fall into the cycling problem (which means the algorithm visits the same part of the solution space repeatedly). Up to now, many efficient strategies have been used to handle this problem [17–20]. Configuration checking (*CC*) [21] strategy is one of those strategies and has been applied to some problems successfully [21–25]. Therefore, we adopt the *CC* strategy to avoid the cycling problem in the local search.

Before introducing the *CC*, we shall give the concept of vertex state. The vertex state of a vertex v indicates whether v is located in the current solution. We can use a Boolean value 1 to represent that v is in the current solution and 0 to represent that v is not in the current solution. The configuration of a vertex v, which can be denoted by an n-dimensional Boolean vector c_v (where n is the number of neighbor vertices of v), is the states of all its neighbor vertices.

The main idea of the *CC* strategy is that a vertex v is forbidden to add back to the current solution if its configuration keeps unchanged after it was removed from the current solution last time. This strategy is intuitive and reasonable in avoiding cycling problem, as it prevents the search from facing the same scenario again. For example, for a $v \in C$, suppose its configuration is $c_{v0} = (0,1,0,1)$ after removing v out of C, then after several steps of searching, v is selected again according to the greedy function value and suppose its configuration is c_{v1} now. When $c_{v0} = c_{v1}$, i.e., configuration not changed, if v is added back to the C, then the search goes back to the same situation before v

```
1  initialize each element value of Change array to 1;
2  C* = C;
3  while true do
4     if C is a vertex cover then
5        if C is connected then
6           if |C| < |C*| then
7              C* = C;
8           end
9        else
10          return C*;
11       end
12       drop a vertex u with the highest score from C and update the Change array;
13       continue;
14    end
15    choose an uncovered edge e randomly;
16    choose a vertex v ∈ e such that Change[v] = 1 with a higher score, C = {v} ∪ C and update the Change array;
17 end
```

Algorithm 3: LocalSearch(C).

was removed out of C in last time and the same solution space will be searched repeatedly. However, this situation will not occur when $c_{v0} \neq c_{v1}$, i.e., configuration changed (e.g., $c_{v1} = (1, 1, 0, 1)$). A Boolean array *Change* is used to implement the *CC* strategy. The element in the array *Change* is *Change*[v], which denotes whether the configuration of vertex v is changed after it was removed from the current solution last time. We use *Change*[v] = 1 to represent that the configuration of vertex v is changed, and *Change*[v] = 0 to represent that the configuration of vertex v is not changed. Only the vertex v whose *Change*[v] = 1 is allowed to be added to the current solution in the local search process. In the process of local search, the values of *Change* are updated according the rules below [21].

(i) Rule 1: In the beginning, for each vertex v, set *Change*[v] to 1.

(ii) Rule 2: When removing v from C, reset *Change*[v] to 0.

(iii) Rule 3: When u changes its state, for each $v \in \mathrm{N}(u) \backslash \mathrm{C}$, *Change*[$v$] is set to 1.

3.2.2. Local Search Procedure. In this subsection, the local search procedure is discussed at length. The main steps of the LocalSearch are listed in Algorithm 3.

The local search procedure performs as follows. In the first place, all elements of the *Change* are initialized to 1 (line 1), which means that all vertices are allowed to be added to the current solution in the beginning. Next, the local optimal solution C^* is initialized as C, where C is generated by the construction phase (line 2). Then, in the loop (lines 3–17), the LocalSearch(C) checks the feasibility of the current solution C. If C is a *CVC* solution which has a smaller vertex number than C^*, then C^* will be updated by C and a vertex v with the highest *score* will be removed from C, and then the procedure starts the next cycle (lines 4–14). If C is a *VC* but not a *CVC* anymore after v is removed, which means the procedure reaches a local optimal solution, then C^* will be returned as the final result of the local search procedure (lines 9–11). If C is not a *VC* anymore after v is removed, the procedure chooses an uncovered edge e randomly and then chooses a vertex $v \in e$ such that *Change*[v]=1 with a higher *score*, and adds to $C = \{v\} \cup C$ (lines 15-16). The *Change* array is updated according to Rule 2 and Rule 3 when a vertex changes its state (lines 12,16).

3.3. Time Complexity Analysis. In this subsection, we discuss the time complexity of the main components of the GRASP-CVC algorithm.

First, we analyze the process for constructing the initial solution in the algorithm. In each loop, we need to update the *score* for $|V|$ vertices, and scan $|N(C)|$ vertices in order to update the *RCL*. If every time we add a vertex to C, d edges are covered on average. Then, we can get a solution in $O(|E|/d * (|V| + |N(C)|)) = O(|E| * |V|)$.

Next, we analyze the time complexity of the local search algorithm. In this part, there are three operations that affect the running time of the algorithm: *dropping a vertex from* C (Algorithm 3, line 12), *choosing an uncovered edge* (Algorithm 3, line 15), and *updating the Change array* (Algorithm 3, lines 12 and 16). Since the number of vertexes in C is $|C|$, the first operation can be done in a time of $O(|C|)$. In our implementation, we maintain a set to record uncovered edges, so an uncovered edge can be chosen in $O(1)$. The time complexity of the third operation depends on the degree of the operated vertex, and therefore this work can be done in $\mathrm{O}(\delta)$, where δ is the maximum degree of the vertices. Thus, the time complexity of the local search is $O(|C| + 1 + \delta) = O(|C| + \delta)$.

Overall, the run-time complexity of the GRASP-CVC is $O(|E| * |V| + |C| + \delta)$.

4. Computational Experiments

During this section, the effectivity and efficiency of the GRASP-CVC algorithm are evaluated by performing some

```
1  initialize the population P with PopSize different individuals and compute the fitness value of each individual;
2  C* = select one individual with minimum fitness value from P;
3  while not reach the cutoff time do
4      Sort individuals in P according to fitness in ascending order;
5      Select the top EliteNum individuals as elite individuals to pass directly to the next generation;
6      for step = 1 to (PopSize − EliteNum)/2 do
7          Select two individuals Ind_1 and Ind_2 through the tournament selection method in the remaining individuals;
8          Perform crossover operations on Ind_1 and Ind_2 with probability CroProb to generate two new individuals Ind'_1 and Ind'_2;
9          Perform mutation operations on Ind'_1 and Ind'_2 with probability MutProb;
10     end
11     C = an individual with the minimum fitness value in the new population P;
12     if fitness(C) < fitness(C*) then
13         C* = C;
14     end
15 end
16 return C*;
```

Algorithm 4: GA (*PopSize*, *EliteNum*, *CroProb*, *MutProb*, *cutoff*, and *seed*).

comparison experiments. We compare the GRASP-CVC algorithm with genetic algorithm (GA) for CVC designed by us and the current best approximate algorithm (2-approximation algorithm) we know in general graphs [12]. Genetic algorithm for CVC is an evolutionary algorithm using techniques inspired by natural evolution. Since GA has achieved good results in other combinatorial optimization problems, we compare our algorithm GRASP-CVC against GA. The main steps of the GA for CVC are listed in Algorithm 4 and the algorithm performs as follows. In the first place, to initialize the population P, $PopSize$ different individuals are generated randomly. Each individual is a sequence of 0, 1 of length n, where n is the number of vertices in the graph, 1 indicates that the corresponding vertex is in the solution, and 0 indicates that it is not in the solution. The fitness value of each individual is calculated while generating the individual (line 1). The fitness value is the number of 1 in the sequence, and the smaller the fitness, the better the solution. For example, if both individuals ind_1(001010) and ind_2(100011) are feasible solutions, the fitness values of ind_1 and ind_2 are 2 and 3, respectively, and the solution ind_1 is better than the solution ind_2. Next, select an individual with minimum fitness value from P as the current best solution C^* (line 2). Then, in main loop (lines 3–15), the algorithm sorts the individuals in P according to the fitness value in ascending order and selects $EliteNum$ individuals as the elites to pass directly to the next generation (lines 4,5). To generate ($PopSize - EliteNum$) new individuals, the *for* loop is iterated ($PopSize - EliteNum$)/2 times (lines 6–10). In each iteration of the *for* loop, two individuals ind_1 and ind_2 are selected from the remaining individuals using tournament selection method firstly (line 7). Then, crossover operation is performed on ind_1 and ind_2 with probability $CroProb$ to get two new individuals ind'_1 and ind'_2 (line 8). When two individuals perform the crossover operation, the algorithm first randomly selects a position in the sequence and then exchanges the sequence after the position. For example, if the two individuals are ind_1(011010) and ind_2(100011), respectively, and the crossover position is the fourth position, then the two new individuals after the crossover operation are ind'_1(011011) and ind'_2(100010). Next, mutation operation is performed on ind'_1 and ind'_2 with probability $MutProb$ (line 9). When performing the mutation operation, the algorithm randomly selects a position in the sequence and flips the value at that position; that is, 1 becomes 0, and 0 becomes 1. After the crossover and mutation operations, if the individual is no longer a feasible solution, each time we randomly change a 0 in the sequence to 1 until the individual becomes a feasible solution again. Individuals obtained in the above operations form a new population P. If the individual with the minimum fitness in the P is better than current best solution C^*, C^* is updated with C (line 11–14). Finally, the algorithm returns the optimal solution C^* found after the time is exhausted (line 16). Owing to the fact that the researches on CVC problem mainly focused on theoretical studies, there is no available approximation *CVC* solver, so we implement the 2-approximation algorithm proposed in [12]. We implement the algorithms GRASP-CVC, GA, and the 2-approximation algorithm in the C++ programming language. All of the experiments are carried on a work station under windows 7 operating system, 3.30GHZ CPU and 8GB memory.

4.1. Benchmark Instances. In the experiments, we choose two well-known benchmarks in the field of *MVC* research, the DIMACS and BHOSLIB instance sets. DIMACS benchmark contains both structured and randomized instances. The structured instances are generated from practical problems, like coding theory, Keller conjecture, and so on. The randomized instances are generated from the stochastic models, such as brock instances. The scale of these problem instances is from 50 vertices and 1000 edges to more than 5500 vertices and 5 million edges. BHOSLIB benchmark is famous for its hardness. The benchmark instances are transformed from *SAT* instances that are generated in the phase transition area. And the instances in the phase transition area have

been proved hard. From the two benchmarks, we select 37 DIMACS benchmark instances and 40 BHOSLIB benchmark instances, which are all employed in the best *MVC* solver [25].

4.2. Experiment Parameter Settings. Before reporting the experimental results, we shall introduce some parameter settings.

(i) GRASP-CVC: There are two main parameters: maximum number of iterations (*MaxIter*) and random number (random *seed*). According to our experimental experience, we set *MaxIter* to 5000 and random seed to an interval from 1 to 10. This is because in order to evaluate the robustness of GRASP-CVC, we executed ten times for each instance and for each time used different random seeds.

(ii) GA for CVC: There are six parameters: population size (*PopSize*), the number of elite individuals (*EliteNum*), crossover probability (*CroProb*), mutation probability (*MutProb*), cutoff time (*cutoff*), and random number (random *seed*). In our experiments, we set the first five parameters to 20, 10, 0.85, $1/n$ (n is the number of vertices of the input instance), and 1000 seconds, respectively. And for the same consideration as GRASP-CVC, we set different random seeds (1 to 10) in ten times running.

In addition, for the sake of comparison between GRASP-CVC and GA for CVC, we also set a cut-off time to 1000 seconds to the two algorithms.

4.3. Experimental Results. In this subsection, we will provide the computational results of GRASP-CVC (GRASP-CVC) GA for CVC (GA) and the 2-approximation algorithm (2-Aprox) on the two chosen benchmarks. In the results, we provide the following information: the number of vertices and edges of each instance ($|V|, |E|$), the best known size of the *MVC* solution (*MVC*), the best solution size solved by the corresponding algorithms (*best*), the average size of solutions solved by the corresponding algorithms (*avg*), and the average time consumed by the corresponding algorithms (*time*). In addition, the *MVC* size with star ($*$) has been proved to be the optimal size of *MVC* solution.

In Table 1, we present the experimental results on DIMACS benchmark. As is shown in the table, GRASP-CVC finds the better quality CVC than GA and 2-Aprox on all the 37 DIMACS instances. On many instances, the solutions found by GRASP-CVC are very close to the optimal MVC and it even finds the solutions of the same size as optimal MVC on 10 instances in a very short time, which means it finds the optimal CVC solutions on these 10 instances. Besides, the GRASP-CVC consumes less time compared to GA on most of the instances, which indicates that the GRASP-CVC is very efficient. Moreover, the GA fails to find a solution within the cut-off time on several instances (C2000.9.mis, C2000.5.mis, C4000.5.mis, and so on), so the column for GA is marked as "N/A". Another point worth noting is that though the 2-Aprox takes less time compared to the other algorithms, its solutions are not so satisfactory.

Table 2 provides the results on BHOSLIB benchmark. We can get the same conclusion as on DIMACS that the GRASP-CVC performs better than the other two algorithms on BHOSLIB instances, even though it takes much more time than 2-Aprox. The solutions found by GRASP-CVC are close to the optimal MVC and the columns *best* and *avg* of GRASP-CVC have the same values on almost all of the instances, which means that the GRASP-CVC gets almost solutions of the same size in each of the 10 times of running, and this also demonstrates the stability of the GRASP-CVC.

The comparative and experimental analyses above show that the GRASP-CVC possesses very good effectiveness and efficiency for CVC problem. It performs better than the other two algorithms in solution quality and outperforms the GA whether in solution quality or in time consumption. Moreover, the GRASP-CVC gets almost solutions of the same size in each of the 10 times running, which demonstrates the stability of the GRASP-CVC.

5. Conclusion

In this paper, a heuristic algorithm GRASP-CVC for connected vertex cover problem was proposed. A greedy function and a restricted candidate list (*RCL*) were proposed to help in constructing a high quality initial solution. Furthermore, the configuration checking (*CC*) strategy was employed to reduce the cycling problem and improve the efficiency of the search. Experimental results demonstrate that GRASP-CVC works better than the comparison algorithms, which validates the effectiveness and efficiency of our GRASP-CVC solver. In the future, we will further study various heuristic methods and hope to design a more powerful heuristic algorithm to deal with *CVC*.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was fully supported by the National Natural Science Foundation of China under Grants No. 61370156, No. 61403076, and No. 61763003; Research Fund for the Doctoral Program of Higher Education No. 20120043120017; Program for New Century Excellent Talents in University No. NCET-13-0724; the Large-Scale Scientific Instrument and Equipment Sharing Project of Jilin Province (20150623024TC-03);

Table 1: Experimental results on DIMACS Instances.

Instance	V,E	MVC	2-Aprox			GA			GRASP-CVC		
			time	best	avg	time	best	avg	time	best	avg
*brock*200_2.*mis*	200,10024	188*	<0.01	198	199.7	412.81	194	194	0.04	**190**	**190**
*brock*200_4.*mis*	200,6811	183*	<0.01	195	197.2	790.61	191	191.8	1.4	**184**	**184**
*brock*400_2.*mis*	400,20014	371*	<0.01	396	397.5	736.09	391	391.6	6.2	**376**	**376**
*brock*400_4.*mis*	400,20035	367*	<0.01	396	396.6	947.48	390	391.8	10.29	**376**	**376**
*brock*800_2.*mis*	800,111434	776*	<0.01	798	798.2	523.34	794	794.9	35.04	**780**	**780**
*brock*800_4.*mis*	800,111957	774*	<0.01	798	798.1	382.88	795	795.4	174	**780**	**780**
*C*125.9.*mis*	125,787	91*	<0.01	116	117.4	617.34	103	103.8	<0.01	**91**	**91**
*C*250.9.*mis*	250,3141	206*	<0.01	240	243.2	740.32	230	230.8	2.38	**207**	**207**
*C*500.9.*mis*	500,12418	443*	<0.01	491	494.2	797.98	483	484.9	12.05	**448**	**448**
*C*1000.9.*mis*	1000,49421	932	<0.01	994	995.7	21.39	987	989.1	66.33	**939**	**939**
*C*2000.9.*mis*	2000,199468	1920	<0.01	1992	1992.3	N/A	N/A	N/A	96.85	**1933**	**1933**
*C*2000.5.*mis*	2000,999164	1984	<0.01	1998	1998	N/A	N/A	N/A	942.59	**1986**	**1986.1**
*C*4000.5.*mis*	4000,3997732	3982	0.03	3998	3998.2	N/A	N/A	N/A	400.23	**3986**	**3986**
*DSJC*500.5.*mis*	500,62126	487*	<0.01	498	499.6	289.4	495	495.7	2.42	**487**	**487**
*DSJC*1000.5.*mis*	1000,249674	985*	<0.01	998	999.8	585.69	996	996.7	887.52	**986**	**986**
*gen*200_*p*0.9_44.*mis*	200,1990	156*	<0.01	189	192.8	796.34	179	180.1	0.06	**164**	**164**
*gen*200_*p*0.9_55.*mis*	200,1990	145*	<0.01	186	191.5	675.44	177	179.6	0.06	**156**	**156**
*gen*400_*p*0.9_55.*mis*	400,7980	345*	<0.01	388	391.5	772.41	382	382.8	72.48	**358**	**358**
*gen*400_*p*0.9_65.*mis*	400,7980	335*	<0.01	389	390.9	789.76	383	383.4	18.81	**354**	**354**
*gen*400_*p*0.9_75.*mis*	400,7980	325*	<0.01	391	393.1	782.04	382	382.7	2.8	**357**	**357**
*hamming*8 − 4.*mis*	256,11776	240*	<0.01	255	255.9	478.87	248	249.3	0.02	**240**	**240**
*hamming*10 − 4.*mis*	1024,89600	984*	<0.01	1023	1023.5	613.39	1018	1018.8	273.01	**990**	**990**
*keller*4.*mis*	171,5100	160*	<0.01	170	170.1	839.79	163	163.7	3.93	**160**	**160**
*keller*5.*mis*	776,74710	749*	<0.01	775	775.6	272.85	768	770.7	4.64	**756**	**756**
*keller*6.*mis*	3361,1026582	3302	<0.01	3360	3360.1	N/A	N/A	N/A	82.16	**3324**	**3324**
*MANN_a*27.*mis*	378,702	252*	<0.01	290	293.8	621.91	281	283.9	0.05	**260**	**260**
*MANN_a*45.*mis*	1035,1980	690*	<0.01	765	766.4	956.81	830	838.1	0.85	**704**	**704**
*MANN_a*81.*mis*	3321,6480	2221	<0.01	2353	2357.6	878.8	3093	3104	5.10	**2241**	**2241**
*p_hat*300 − 1.*mis*	300,33917	292*	<0.01	298	298	473.69	296	296.1	0.11	**292**	**292**
*p_hat*300 − 2.*mis*	300,22922	275*	<0.01	298	298	516.16	291	291.9	0.64	**275**	**275**
*p_hat*300 − 3.*mis*	300,11460	264*	<0.01	294	296	691.7	287	288.1	0.34	**264**	**264**
*p_hat*700 − 1.*mis*	700,183651	689*	<0.01	698	699.8	155.67	697	697.3	7.78	**689**	**689**
*p_hat*700 − 2.*mis*	700,122922	656*	<0.01	700	700	95.72	689	691.3	1.79	**656**	**656**
*p_hat*700 − 3.*mis*	700,61640	638*	<0.01	694	695.9	30.67	688	688.9	176.31	**638**	**638**
*p_hat*1500 − 1.*mis*	1500,839327	1488*	<0.01	1500	1500	N/A	N/A	N/A	19.73	**1489**	**1489.1**
*p_hat*1500 − 2.*mis*	1500,555290	1435*	<0.01	1498	1498.1	N/A	N/A	N/A	27.61	**1438**	**1438**
*p_hat*1500 − 3.*mis*	1500,277006	1406	<0.01	1496	1496	N/A	N/A	N/A	210.48	**1409**	**1409**

Table 2: Experimental results on BHOSLIB Instances.

Instance	V,E	MVC	2-Aprox			GA			GRASP-CVC		
			time	best	avg	time	best	avg	time	best	avg
*frb*30 − 15 − 1.*mis*	450,17827	420*	<0.01	449	449.6	996.69	439	440.2	11.37	**424**	**424**
*frb*30 − 15 − 2.*mis*	450,17874	420*	<0.01	447	448.4	826.6	438	440.2	7.87	**425**	**42**5
*frb*30 − 15 − 3.*mis*	450,17809	420*	<0.01	449	449.7	915.5	438	440.2	13.43	**424**	**424**
*frb*30 − 15 − 4.*mis*	450,17831	420*	<0.01	448	449.3	885.97	439	439.9	15.24	**424**	**424**
*frb*30 − 15 − 5.*mis*	450,17794	420*	<0.01	448	449.3	593.41	439	440.2	25.61	**423**	**423**
*frb*35 − 17 − 1.*mis*	595,27856	560*	<0.01	592	593.7	890.5	584	586.1	29.45	**565**	**565**
*frb*35 − 17 − 2.*mis*	595,27847	560*	<0.01	592	592.8	801.11	585	586.1	8.54	**565**	**565**
*frb*35 − 17 − 3.*mis*	595,27931	560*	<0.01	592	592.7	850.7	583	585.2	2.7	**565**	**565**
*frb*35 − 17 − 4.*mis*	595,27842	560*	<0.01	592	593.6	868.33	585	586.1	172.98	**565**	**565**
*frb*35 − 17 − 5.*mis*	595,28143	560*	<0.01	594	594.5	688.65	585	585.9	34.48	**565**	**565**
*frb*40 − 19 − 1.*mis*	760,41314	720*	<0.01	758	759	897.75	750	751.7	187.74	**728**	**728**
*frb*40 − 19 − 2.*mis*	760,41263	720*	<0.01	757	758	22.05	750	751.2	213.66	**726**	**726**
*frb*40 − 19 − 3.*mis*	760,41095	720*	<0.01	758	759.2	797.46	749	751.3	101.08	**725**	**725**
*frb*40 − 19 − 4.*mis*	760,41605	720*	<0.01	757	758.1	920	749	751.5	8.39	**726**	**726**
*frb*40 − 19 − 5.*mis*	760,41619	720*	<0.01	758	759	715.9	751	751.6	33.32	**725**	**725**
*frb*45 − 21 − 1.*mis*	945,59186	900*	<0.01	944	944.4	460.03	935	936.4	665.53	**908**	**908**
*frb*45 − 21 − 2.*mis*	945,58624	900*	<0.01	942	943.3	437.63	936	937.1	83.9	**908**	**908**
*frb*45 − 21 − 3.*mis*	945,58245	900*	<0.01	941	943	65.3	936	937.1	479.14	**908**	**908**
*frb*45 − 21 − 4.*mis*	945,58549	900*	<0.01	941	942.6	597.27	935	936.4	135.22	**907**	**907**
*frb*45 − 21 − 5.*mis*	945,58579	900*	<0.01	943	943.7	847.46	936	937.2	235.15	**909**	**909**
*frb*50 − 23 − 1.*mis*	1150,80072	1100*	<0.01	1146	1147.6	509.18	1141	1142.9	409.1	**1109**	**1109**
*frb*50 − 23 − 2.*mis*	1150,80851	1100*	<0.01	1147	1148.6	86.18	1139	1142	87.47	**1110**	**1110**
*frb*50 − 23 − 3.*mis*	1150,81068	1100*	<0.01	1147	1148.1	110.17	1139	1142	414.19	**1110**	**1110**
*frb*50 − 23 − 4.*mis*	1150,80258	1100*	<0.01	1147	1148.5	94.35	1141	1142.7	431.32	**1110**	**1110**
*frb*50 − 23 − 5.*mis*	1150,80035	1100*	<0.01	1147	1148.3	125.77	1141	1142.3	837.10	**1109**	**1109**
*frb*53 − 24 − 1.*mis*	1272,94227	1219*	<0.01	1269	1270.7	180.82	1263	1265	695.54	**1230**	**1230**
*frb*53 − 24 − 2.*mis*	1272,94289	1219*	<0.01	1269	1270.4	210.55	1259	1263.9	390.44	**1230**	**1230**
*frb*53 − 24 − 3.*mis*	1272,94127	1219*	<0.01	1270	1271.2	173.43	1262	1264.2	552.58	**1229**	**1229**
*frb*53 − 24 − 4.*mis*	1272,94308	1219*	<0.01	1270	1270.4	225.5	1264	1264.9	588.66	**1229**	**1229**
*frb*53 − 24 − 5.*mis*	1272,94226	1219*	<0.01	1270	1270.6	154.67	1263	1264.9	95.67	**1230**	**1230**
*frb*56 − 25 − 1.*mis*	1400,109676	1344*	<0.01	1398	1399.1	226.86	1390	1392.3	90.55	**1357**	**1357**
*frb*56 − 25 − 2.*mis*	1400,109401	1344*	<0.01	1397	1397.3	234.39	1389	1391.7	412.38	**1353**	**1353**
*frb*56 − 25 − 3.*mis*	1400,109379	1344*	<0.01	1397	1398.6	306.88	1391	1392.7	190.64	**1356**	**1356**
*frb*56 − 25 − 4.*mis*	1400,110038	1344*	<0.01	1398	1399	411.44	1392	1392.4	47.17	**1356**	**1356**
*frb*56 − 25 − 5.*mis*	1400,109601	1344*	<0.01	1397	1398.4	262.55	1392	1392.8	616.93	**1355**	**1355**
*frb*59 − 26 − 1.*mis*	1534,126555	1475*	<0.01	1533	1533.2	276.64	1525	1527	656.15	**1487**	**1487**
*frb*59 − 26 − 2.*mis*	1534,126163	1475*	<0.01	1531	1532.5	351.89	1523	1526.2	100.06	**1488**	**1488**
*frb*59 − 26 − 3.*mis*	1534,126082	1475*	<0.01	1531	1532.5	366.51	1526	1527.2	495.53	**1489**	**1489**
*frb*59 − 26 − 4.*mis*	1534,127011	1475*	<0.01	1531	1532.4	300	1525	1525.9	319.41	**1487**	**1487**
*frb*59 − 26 − 5.*mis*	1534,125982	1475*	<0.01	1533	1533.3	258.46	1525	1527.2	662.72	**1487**	**1487**

The Natural Science Foundation for Youths of Jilin Province (20160520104JH).

References

[1] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[2] H. Moser, *Exact algorithms for generalizations of vertex cover [M.S. thesis]*, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, 2005.

[3] Z. Zhang, X. Gao, and W. Wu, "PTAS for connected vertex cover in unit disk graphs," *Theoretical Computer Science*, vol. 410, no. 52, pp. 5398–5402, 2009.

[4] P. Guo, J. Wang, X. H. Geng, C. S. Kim, and J.-U. Kim, "A variable threshold-value authentication architecture for wireless mesh

networks," *Journal of Internet Technology*, vol. 15, no. 6, pp. 929–935, 2014.

[5] J. Shen, H. Tan, J. Wang, J. Wang, and S. Lee, "A novel routing protocol providing good transmission reliability in underwater sensor networks," *Journal of Internet Technology*, vol. 16, no. 1, pp. 171–178, 2015.

[6] P. L. K. Priyadarsini and T. Hemalatha, "Connected vertex cover in 2-connected planar graph with maximum degree 4 is NP-complete," *International Journal of Mathematical, Physical and Engineering Sciences*, vol. 2, no. 1, pp. 51–54, 2008.

[7] H. Fernau and D. F. Manlove, "Vertex and edge covers with clustering properties: complexity and algorithms," *Journal of Discrete Algorithms*, vol. 7, no. 2, pp. 149–167, 2009.

[8] T. Watanabe, S. Kajita, and K. Onaga, "Vertex covers and connected vertex covers in 3-connected graphs," in *Proceedings of the IEEE International Sympoisum on Circuits and Systems*, vol. 2, pp. 1017–1020, Singapore, Singapore, 1991.

[9] D. Mölle, S. Richter, and P. Rossmanith, "Enumerate and expand: improved algorithms for connected vertex cover and tree cover," *Theory of Computing Systems*, vol. 43, no. 2, pp. 234–253, 2008.

[10] D. Binkele-Raible, *Amortized Analysis of Exponential Time-and Parameterized Algorithms: Measure & Conquer and Reference Search Trees*, Praca doktorska, University of Trier, Trier, Germany, 2010.

[11] C. Savage, "Depth-first search and the vertex cover problem," *Information Processing Letters*, vol. 14, no. 5, pp. 233–235, 1982.

[12] T. Fujito and T. Doi, "A 2-approximation NC algorithm for connected vertex cover and tree cover," *Information Processing Letters*, vol. 90, no. 2, pp. 59–63, 2004.

[13] B. Escoffier, L. Gourvès, and J. Monnot, "Complexity and approximation results for the connected vertex cover problem in graphs and hypergraphs," *Journal of Discrete Algorithms*, vol. 8, no. 1, pp. 36–49, 2010.

[14] J. Cardinal and E. Levy, "Connected vertex covers in dense graphs," in *APPROX 2008, RANDOM 2008: Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, vol. 5171 of *Lecture Notes in Computer Science*, pp. 35–48, Springer, Berlin, Germany, 2008.

[15] Y. Li, Z. Yang, and W. Wang, "Complexity and algorithms for the connected vertex cover problem in 4-regular graphs," *Applied Mathematics and Computation*, vol. 301, pp. 107–114, 2017.

[16] M. G. C. Resende and C. C. Ribeiro, "GRASP: Greedy randomized adaptive search procedures," in *Search Methodologies*, pp. 287–312, Springer, 2014.

[17] Y. Zhou, H. Zhang, R. Li, and J. Wang, "Two local search algorithms for partition vertex cover problem," *Journal of Computational and Theoretical Nanoscience*, vol. 13, no. 1, pp. 743–751, 2016.

[18] X. Li, J. Zhang, and M. Yin, "Animal migration optimization: an optimization algorithm inspired by animal migration behavior," *Neural Computing and Applications*, vol. 24, no. 7-8, pp. 1867–1877, 2014.

[19] Y. Wang, S. Cai, and M. Yin, "Two efficient local search algorithms for maximum weight clique problem," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 805–811, Phoenix, Ariz, USA, February 2016.

[20] Y. Wang, M. Yin, D. Ouyang, and L. Zhang, "A novel local search algorithm with configuration checking and scoring mechanism for the set *k*-covering problem," *International Transactions in Operational Research*, vol. 24, no. 6, pp. 1463–1485, 2017.

[21] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artificial Intelligence*, vol. 175, no. 9-10, pp. 1672–1696, 2011.

[22] C. Luo, S. Cai, W. Wu, and K. Su, "Double configuration checking in stochastic local search for satisfiability," in *Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI 2014, 26th Innovative Applications of Artificial Intelligence Conference, IAAI 2014 and the 5th Symposium on Educational Advances in Artificial Intelligence, EAAI 2014*, pp. 2703–2709, Québec City, Canada, July 2014.

[23] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su, "CCLS: an efficient local search algorithm for weighted maximum satisfiability," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1830–1843, 2015.

[24] J. Gao, J. Wang, and M. Yin, "Experimental analyses on phase transitions in compiling satisfiability problems," *Science China Information Sciences*, vol. 58, no. 3, pp. 1–11, 2015.

[25] S. Cai, K. Su, C. Luo, and A. Sattar, "NuMVC: an efficient local search algorithm for minimum vertex cover," *Journal of Artificial Intelligence Research*, vol. 46, pp. 687–716, 2013.